# Expressions & interactions
# Conditionals
# Solving a problem

# binary representation (8 bits)

- positives: first bit=0

```
  0 = 00000000
  1 = 00000001
  2 = 00000010
  3 = 00000011
  4 = 00000100
....
127 = 01111111
```

- negatives: first bit=1, stands for -128 "two complement"

```
-128 = 10000000
-127 = -128+1 =10000001
-126 = -128+2 =10000010
-125 = -128+3 =10000011
 ...
...
  -1 = -128+127 =11111111
```

- can also be vied as modulo arithmetic

- range : -128 to 127

# Arithmetic Operators

- `int a=3, b=7,c=10+3;`
- `c=17+5; a=23-c;`
- `int d=c*2;`
- `b=a%2;`
- `a = 5/2;`

# Assignments

- `int a, b, c;`
  - `a=12;b=1;c=3;`
  - `a= myfunction(1,2,3);` //function call
  - `a=b=c=10;` //assigns c=10, then b=10, then a=10

# Integer Division

- `int a1 = 5/2;`
  - results in a1=2
- `double a2=5/2;`
  - results in a2=2
- `double a3= 5.0/2;`
  - results in a3=2.5
- `double a4 = (double)5/2;`
  - results in a4=2.5

# Mathematical Expressions

- `int n0 = 12 - 6 + 3;`
- `int n1 = 12 - 6/3;`
- `int n2 = 3+5+8;`
  - `int n3 = 3+5+8/3;`
  - `int n4 = (3+5+8)/3;`
  - `double n5 = (double)(3+5+8)/3;`

# Precedence, Parenthesis

- `int n6= 6-3*2+7-1;`
- `int n7 = (6-3)*2+7-1;`
- `int n8 = (6-3)*(2+7)-1;`
- `int n9 = 6-3<<2*2+1;`

# Exponential, Logarithm

- no exponents, use the `pow/log` functions
- `double x = pow(5,3);` //results in x=125
- `double y = log(100);` //returns y=4.605…, natural log
  - `double y2=log10(100);` //returns y=2, log base 10
  - `double y3=log2(100);` //returns y=6.64…, log base 2
- `double y4=log10(pow(10,4));` returns ?
- write a log base 5
  - `log5(x) = log2(x)/log2(5);`

# Square Root, other math

- `sqrt(81)= ?`
- `abs` = absolute value
- `sin, cos, tan` = trigonometry functions
- `exp` =exponential, base e
- `fmod` = modulus for doubles

# Overflow, Underflow

- overflow: result of operation is too big for the range of the type
  - `int a=150000, b=150000;`
  - `int c1=a*b; cout<<"c1="<<c1;`
  - `long c2=a*b ; cout <<"  c2="<<c2<<"\n";`
  - `long c3=(long)a* (long) b ; cout <<" c3="<<c3<<"\n";`
- underflow: result of operation is too small for the range of the type
  - `int a=-150000, b=150000;`
  - `int c1=a*b; cout<<"c1="<<c1;`
  - `long c2=a*b ; cout <<"  c2="<<c2<<"\n";`

# Type Casting

```
int t1=150000,t2=150000;
int t3= t1*t2;
long t4= (long)t1*(long)t2;
```

# Compound Operators

- `int a1=5; a1+=4;` //**results in a1=9**
- `int a2=5; a2-=4;` //**results in a2=1**
- `int b=7; b*=a1;` //**results in b=63**
- `int b2=8; b2/=4;` //**results in b2=2**

# Random Numbers

- really, "pseudorandom"
- `y=rand()` generates a random `int` number
  - y=rand()%100 generates a random int between 0 and 99
  - but when program re-executed, same number
- `srand(seed)` reinitializes the random generator with the seed
  - `use seed=time(0)` for a pseudorandom initialization

# setprecision, fixed, setw

- `setprecision` sets the number of digits displayed
  - total digits = before and after the decimal point
  - remains in effect until changed
  - `double d=123.45;`
  - `cout<<setprecision(4)<<d<<"\n";` //displays 123.4'
  - `cout<<setprecision(5)<<12345.78;` //displays 1.2345e+005
- `fixed` forces cout to display digits, not scientific notation
  - `cout<<setprecision(5)<<fixed<<12345.78;` //displays 12345
- `setw` sets the number of characters of the output
  - `cout<<setprecision(6)<<fixed<<setw(10)<<"|"<< 12345.78<<"|";` //displays | 12345.7|

# showpoint, left, right

- `showpoint`= shows trailing zeros
- `left/right` = sets text alignment

# More `cin, cout`

- `getline(cin, x)`
  - input a line into string x
- `c = cin.get()`
  - input a character into char c
- `cin.ignore(n,c)`
  - skip n characters from keyboard, or until char c is found

# Relational Operators

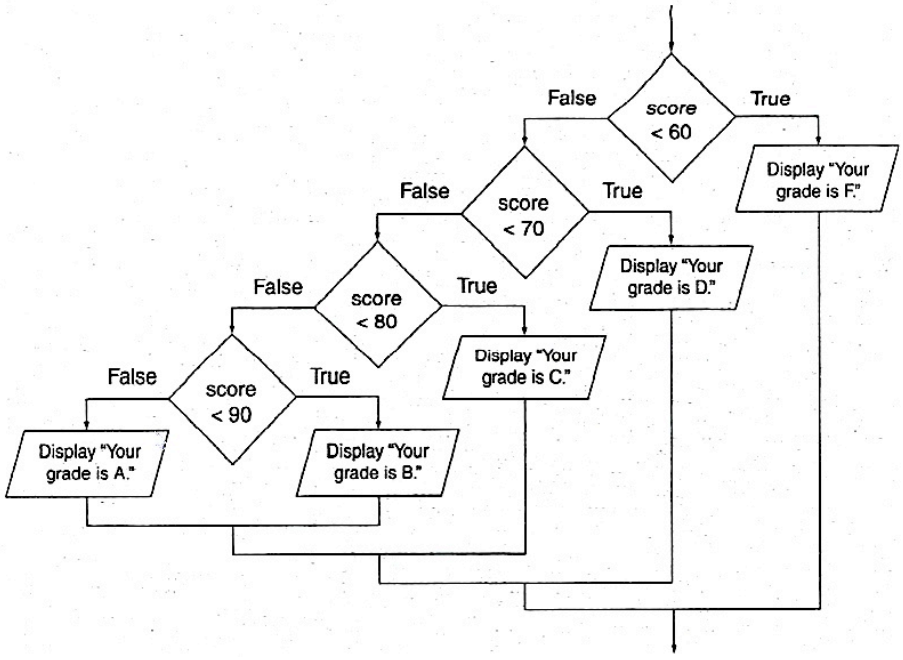
- return bool `true` or `false`
- `x<y, x<=y, x>y, x>=y`
- `x==y, x!=y`
- can write
  - `bool value=(7>5);`
- can write
  - `while(x>y){ ...;}`
- any expression can evaluate to
  - true if its not zero
  - false if it is zero
  - strings evaluate to true, even if they are empty

# if-else

```
if(expression){
   instructions;
}
else{
   different instructions;
}
```

- a decision-making statement
  - `else` is optional
  - curly brackets: they are required if more than one instruction
- the condition/expression is a boolean
  - in fact zero=false, nonzero=true
- branches the code
  - "yes/true/nonzero" branch - a block of instructions is executed
  - "no/false/zero" branch - different block of instructions is executed

# Nested if




# if- else if

```
if (a>0){
  if(b>0){cout << "a>0 and b>0";}
  else if (b<0){cout << "a>0 and b<0";}
    else {cout << "a>0 and b=0";}
}
else if (a<0){
  if(b>0){cout << "a<0 and b>0";}
  else if (b<0){cout << "a<0 and b<0";}
    else {cout << "a<0 and b=0";}
}
else {
  if(b>0){cout << "a=0 and b>0";}
  else if (b<0){cout << "a=0 and b<0";}
    else {cout << "a=0 and b=0";}
}
```

# Logical Operators

- && AND
- || OR
  - short circuit evaluation : first expression evaluation may imply teh second expression is not evaluated
- ! NOT
- XOR?

# Logical Operators

- if (expression1 && expression2) ....
  - if expression1 is false, expression 2 is not evaluated
  - branch "false" executed
- if (expression1 || expression2) ....
  - if expression1 is true, expression 2 is not evaluated
  - branch "true" executed

# Logical Operators

- Precedence (high first)
  - !
  - *, /
  - +, -
  - <, >, ==, <=, >=
  - &&
  - ||
- a=1; b=5;
- !a||b && !b||7>a && !7 >= -1&& !2*5

# Comparing characters, strings

```
char c= 'A';

if (c==65) { cout << "true";}
  else {cout<<"false";}


char d='B';
if (c>d) { cout << "bigger";}
  else {cout<<"smaller";}

if (c>=65 && c<=90) { cout << "capital letter";}
```

# Comparing characters, strings

```
string sx = "ABCD";

string sy = "XBCD";

cout << "\n\nsx="<<sx<<" sy="<<sy<<endl;

if(sx<sy){ cout << "sx<sy";}

if(sx>sy){ cout << "sx>sy";}

if(sx==sy){ cout << "sx==sy";}
```

`strcmp(sx,sy)` : works with strings defined as char* or char[], not `string`

returns negative if `sx<sy`

returns positive if `sx>sy`

returns 0 (zero) if `sx==sy`

# Conditional Operator

- `expression1 ? expression2 : expression3`
- if expression1==true (nonzero) expression2 is evaluated
- otherwise expression3 is evaluated
  - `a= score<80 ? "bad" : "good"`
  - `score>=80 ? a = "good": a="bad";`

# switch

```
switch(expression){
  case constant1:
     instructions;

  case constant2:
     instructions;

   case constant3:
     instructions;
  ....
  default:
     instructions;

}
```

- branching the code
- expression is matched against constants
- when matching, the code executes from the matching case to the end of switch bracket
  - including all following cases
  - use `break` to only execute a case

# Menus using `switch`

- 1.Movies 2.Music 3.Pictures
- Please make a selection (1-3):

```
cin<<selection; switch(selection) {
     case 1:
        do some things with Movies;
     break;
     case 2:
        do some things with Music;
     break;
     case 3:
        do some things with Pictures;
     break;
}
```

# Blocks and scope

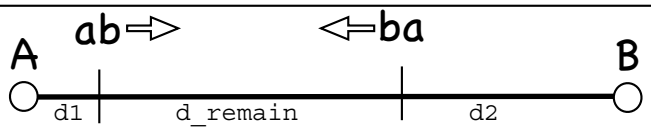
- all variables are defined with a scope
  - can be global
  - the variable can be used only when "in scope"
- usually the scope is within the curly brackets (C++ blocks)
- **VERY BAD IDEA: different scope variables can have same name, compiler wont complain
- whats wrong here:

```
int main{
  int a=5;
  if (a>3){
    int b= 17;
  }
  b=b+1;
  return 0;
}
```

# Solving a problem

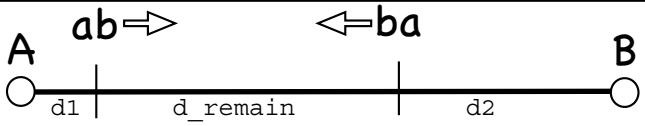
- 2 cities A and B and two trains
  - d = distance between A and B
  - train ab departing from A to B at time t1, speed s1
  - train ba departing from B to A at time t2, speed s2
- Task: find out if the train intersect each other, where, and at what time

# Solving a problem - solution




- at any time t(variable), write down the distance between trains
  - ab has traveled distance d1 = (t-t1)*s1, if 0<=d1<=d
  - ba has traveled distance d2 = (t-t2)*s2, if 0<=d2<=d
  - distance remaining between d_remain = d- d1 -d2
- solve for t, equation d_remain==0
  - d-(t-t1)*s1-(t-t2)*s2 = 0
  - t(s1+s2) = d + t1*s1 + t2*s2
  - compute t = (d+t1*s1 +t2*s2) / (s1+s2)
- with t computed, verify that they actually intersect
  - d1 = (t-t1)*s1; verify 0<=d1<=d
  - d2 = (t-t2)*s1; verify 0<=d2<=d
  - output time t, distance d1 from A, distance d2 from B

# Solving a problem - pseudocode




- input and validate d, s1, t1, s2, t2
- t = (d+t1*s1 +t2*s2) / (s1+s2)
- d1 = (t-t1)*s1; verify 0<=d1<=d
- d2 = (t-t2)*s1; verify 0<=d2<=d
- output time t, distance d1 from A, distance d2 from B

# Solving a problem - coding

```
//input
cout<<"distance"; cin>>d;
cout <<"time departure for train ab (A to B)"; cin>>t1;
cout <<"speed for train ab (A to B)"; cin>>s1;
cout <<"time departure for train ba (B to A)"; cin>>t2;
cout <<"speed for train ba (B to A)"; cin>>s2;

//validate
if (d<=0 || t1<=0 || t2<=0 || s1<=0 || s2<=0)  {
    cout<< "invalid input"; return 0;}

//compute
t = (d+t1*s1 +t2*s2) / (s1+s2);
d1=(t-t1)*s1;
d2=(t-t2)*s2;

//verify, output
if (d1<0 || d1>d) {cout <<"do not intersect"; return 0;}
if (d2<0 || d2>d) {cout <<"do not intersect"; return 0;}
cout <<" trains intersect at time="<<t<<" dist form A="<<d1;
cout <<" dist from B="<<d2<<endl;
return 0;
```